THE BELL SYSTEM TECHNICAL JOURNAL
SAFEGUARD SUPPLEMENT
Printed in U.S.A.

SAFEGUARD Data-Processing System:

System Error Control

By L. J. GAWRON

(Manuscript received January 3, 1975)

Errors occur even in well-designed, well-tested systems. This paper describes how errors are detected and controlled in the SAFEGUARD system and makes recommendations pertaining to the design of error control in large-scale, real-time control systems.

I. INTRODUCTION

SAFEGUARD is a *fault-tolerant* system. It can perform its tactical function even in the presence of many types of errors, including latent design errors, hardware failures, and operator mistakes. This paper describes some of the automatic error-control features of a generic SAFEGUARD Data-Processing System (DPS) and also the important role of manual control in maintaining the operational integrity of the DPS.

II. AVAILABILITY-RELIABILITY REQUIREMENTS

What are the availability and reliability requirements of the SAFEGUARD system? How are they satisfied? What is the role of error control?

As it pertains to SAFEGUARD, availability is the probability that the system is capable of performing its tactical functions—surveillance, tracking, intercept, etc.—at any given point in time. Reliability is the conditional probability that the system will function through the duration of a missile attack provided that the system is available at the beginning of that attack. The product of availability times reliability is required to be high to provide adequate assurance that the system can, at any time, quickly detect a missile attack and successfully defend against it. During peacetime operation, the emphasis is on availability so that the system can perform continuous surveillance and be ready at all times to wage battle against offensive missiles. During a battle, the emphasis is on reliable operation which includes

avoiding significant interruption of tactical performance for any reason, even in response to errors.

Availability and reliability are both enhanced through the use of highly reliable, individual, hardware and software components, as well as through the use of inherently fault-tolerant hardware and software systems. For example, the DPS hardware design features extensive component redundancy and multiprocessor control. (The availability and reliability advantages of multiprocessor computers are commonly accepted today.[1]) The software design also has many features that minimize its vulnerability to errors. For example, it has decentralized system control. This means that total control is not contained in any single, and thus highly vulnerable, software module. It has distributed software execution control, i.e., all processors are treated equally. There is no single controlling processor, which would have an inherently greater vulnerability to errors. Also, the software makes minimal use of particularly vulnerable data structures such as linked lists. In addition to the use of highly reliable components and a fault-tolerant design, thorough testing is also performed to ensure that all components, as well as the total system itself, function as intended.* Thus, error prevention is one of the principal means of satisfying the availability-reliability requirements of the system. The other is error control.

Error control enhances system availability by aiding in rapid detection and replacement of faulty components. The DPS contains redundant components and, in conjunction with the software, it is self-diagnosing. The DPS is normally configured into two distinct partitions: one, called the green partition, is the primary computer system; the other, called the amber partition, is a secondary computer system containing the redundant units. When a faulty green partition unit is detected, a reorganization or reconfiguration of the DPS may be initiated either by the DPS itself or manually by a DPS operator in order to replace the faulty unit with its redundant counterpart. However, such replacements generally require interruption of tactical performance for several seconds.

Error control also enhances reliability by confining errors to minimize their effect on tactical performance, and thus minimize the need for such replacements during a battle. The remainder of this paper describes in greater detail how error control helps to satisfy SAFEGUARD's availability-reliability requirements, especially as they apply to the DPS.

---

* Software-debugging and system-testing methods are described in Refs. 2 and 3.

III. SYSTEM ERROR-CONTROL STRUCTURE

How are errors detected in the SAFEGUARD system? How are the effects of errors confined? How does the system recover from errors? This section discusses the general approach to solving these problems. The following two sections describe in more detail the two principal aspects of error control, namely error detection and error response.

Figure 1 illustrates the basic system error-control structure. Errors may be detected by hardware, by software, or by the DPS operators. Software detections include hardware-reported errors. Likewise, manual detections include both hardware- and software-reported errors.

Software provides the principal responses to hardware and software errors. There are two principal classes of error responses: local responses and system responses. Local responses are attempts to confine or correct errors at the point of detection. System responses replace faulty hardware or software components and restore basic system

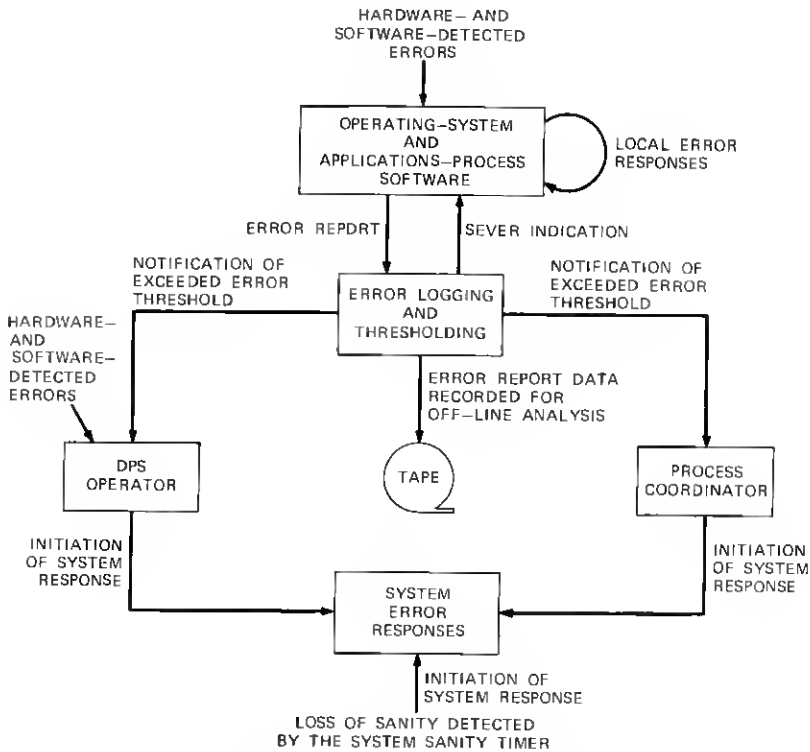

Fig. 1—System error-control structure.

sanity. System responses generally require a brief (several-second) interruption of tactical operation.

During normal peacetime operation, both local and system responses contribute to system availability by correcting errors and replacing faulty components. During battle-mode operation, the emphasis is on local responses to assure reliable operation by confining and correcting errors and to avoid the need to interrupt tactical operation for the purpose of performing system responses.

Specific local responses depend on the type of error detected. Several examples of such responses are described in Section 5.1. In addition to any specific response that might be performed, one common local response is to report the error to a centralized error logging and thresholding function. This function logs (records) the error-report data onto tape for use in off-line error analysis. It also keeps a record of error occurrences. If a report causes an error count or an error rate for the associated class of errors to exceed a prespccified threshold, then several additional common local responses may be taken. One such response is to return a sever indication to the program that reported the error. Severing is a method by which a program is permitted to degrade the operation of certain noncritical parts of the SAFEGUARD system by simply removing them from service. Its purpose is to avoid recurrence of errors. Typical components that could be severed are operating-system modules, such as data recording, or certain CLC peripherals such as printers, tape units, TTYs, etc. In addition to severing, another common local response to an exceeded error threshold is to notify a DPS operator and/or the highest-level software-control function called the process coordinator.* Either may then initiate a system response.

In general, system error responses may be invoked manually, by the process coordinator, or by a special hardware device called the system sanity timer. (Use of the sanity timer is described in Section 4.1.) System responses involve reinitializing the software and/or reconfiguring the DPS to remove faulty components. One of the principal system responses is DPS recovery which includes both DPS reconfiguration and software reinitialization. System error responses are discussed in greater detail in Section 5.2.

IV. ERROR DETECTION

4.1 Hardware detection

Error-detection circuitry is an integral part of the DPS. For example, the processors detect errors such as arithmetic overflow or attempts

* The entire collection of operating system and application software that execute on a single CLC partition is called a process.

to store data into nonexistent memory locations. When such errors are detected, a processor interrupt is generated and the processor transfers execution control, via the operating system, to the program's local-level interrupt-response code. Peripherals detect various types of input/output (I/O) errors, e.g., data-transfer parity errors. Such errors are reported to the software via I/O status returns.

In addition to the error-detection logic, which is a part of basic circuit design, the DPS also contains hardware devices specifically designed to aid in error detection. One such device is the CLC's status unit. It reflects the hardware status of each processor, memory rack, and peripheral, as well as of the radar and missile equipment. This status information obtained from the hardware is accessible to the software and displayed to the operators. Typical status-unit indicators are "processor disabled," "tape unit power marginal," "missile equipment internal error," etc.

Another special error-detection device is the Maintenance and Diagnostic Subsystem (M&DSS) sanity timer. This timer must be reset by the operating system's task scheduler every 50 ± 10 ms as an indication of basic system sanity, i.e., that the software is still executing on the CLC. If the operating system fails to reset it within the correct time interval, the sanity timer will automatically initiate DPS recovery.

4.2 Software detection

Just as error-detection circuitry is an integral part of the hardware, error-detection code is an integral part of the software. For example, the operating system performs input-validity checks on call parameters and the weapons process performs data-reasonableness checks on important data such as radar return signals.

The software also performs several types of hardware diagnostic tests. The operating system performs diagnostics on the DPS equipment; the weapons process performs diagnostics on the radar and missile equipment. For example, whenever the operating system reconfigures the DPS, it performs normal path diagnostics to verify that each green-partition CLC unit functions properly. Also, during tactical execution, CLC units and peripherals in both partitions undergo additional tests. For example, the operating system contains programs called real-time exercisers which test each green-partition memory rack every five minutes. They compare the entire program-store contents with a program-store image on disc to verify that no programs have been modified. They "read test" each variable store rack in its entirety, and they "write test" the first two words and the last two words of each variable store rack by storing test-pattern data into these words and then fetching the words to verify their contents. These four

words in each variable store rack are reserved for this testing purpose. The weapons process contains continuously running radar tests that verify the basic functional operation of the radars. It also contains manually invokable radar tests and missile tests, which are more extensive diagnostics and which are used when faults are suspected in this equipment.

Extensive M&DSS diagnostics, capable of isolating faults to the chassis level, are also performed on amber CLC units and peripherals. All DPS units are periodically reconfigured out of the green partition (replaced by their redundant counterparts) in order to undergo such testing in the amber partition. The purpose of these tests is to minimize the probability of failure in green-partition units by detecting potentially faulty units before they actually fail. M&DSS tests are scheduled by the CLC operating system and are initiated manually. Processors may be reconfigured without terminating execution and are scheduled for M&DSS testing hourly. Other CLC units and the I/O subsystem require an interruption of tactical execution in order to be reconfigured. The entire I/O subsystem is scheduled for M&DSS testing every four hours. CLC units other than processors are not automatically scheduled for M&DSS testing; however, such tests may be initiated on those units manually at any time.[4]

In addition to hardware diagnostic tests, a system exerciser[3] is used to periodically test much of the total hardware/software system.

4.3 Hardware- and software-reported errors

The hardware and the software report many of the errors they detect to the DPS operators. For example, the operators' consoles have many hardware- and software-controlled error-indicator lamps. A system-status panel displays much of the information in the CLC's status unit, thus indicating the operational status (working, faulted, off-line, etc.) of the CLC units and peripherals. Software also notifies the operators of exceeded error thresholds via error-report messages. With the wide variety of error-status information available to him, a DPS operator often better comprehends the system's error environment than do either the hardware or the software and, in many cases, he must determine whether or not a system level response should be initiated.

V. ERROR RESPONSES

5.1 Local responses

Local error responses are attempts to automatically confine or correct errors at the point of detection. They are important in all modes of operation, but especially in the battle mode where they are a significant factor in short-term system reliability.

Programs commonly use the centralized error-logging-and-thresholding function to report, record, and threshold errors they detect. They also perform many kinds of specific local responses designed to correct or confine the effects of a specific type of error detected. The following are several typical examples of such responses.

A program's response to a processor interrupt might be to reinitialize a critical portion of its data base using default values, to unlock any locked data sets, and to exit. If an I/O error is detected, a program might retry the I/O operation. If a radar return-tracking signal fails a data-reasonableness check, a program might employ an algorithm to "coast" the object's track for one radar cycle.

Suppose repeated error indications in the status unit for a peripheral device cause an error-report threshold to be exceeded. If the peripheral is not essential for tactical operation, the peripheral device manager could sever it, thereby degrading system operation but avoiding recurrence of the errors and also avoiding the possibility of propagating the errors into other parts of the system.

In the case where memory errors detected and reported by the real-time exercisers exceed a threshold for a certain memory rack, the only local response is the error-logging-and-thresholding function's notification to a DPS operator and to the process coordinator. Either may then initiate a system response to replace the rack with a spare. Such a replacement might be done during surveillance-mode operation, but not during a battle. During battle-mode operation, the software's local responses must be able to recover from any errors that might occur either in the memories or in other parts of the system.

5.2 System responses

System level error responses are used to reinitialize the system or to replace faulty components. They are invoked automatically by the system sanity timer or by the process coordinator in response to certain errors that cannot be easily corrected at the local level. In many instances, they are invoked manually in response to errors or combinations of errors reported by the hardware or the software. System responses are performed by the operating system but they are never initiated by it. System-error responses contribute to system availability, but they may be inhibited during a battle to prevent interruption of tactical operation.

There are three basic system level error responses: reinitialization, reconfiguration, and DPS recovery. Reinitialization involves reloading the system's entire data base. It can be initiated by the process coordinator to restore severed software components. Reconfiguration involves swapping DPS units between the green and amber partitions. It provides a method for the software's process coordinator or for an

operator to replace faulty or severed hardware units in the tactical (green) partition with their redundant counterparts from the amber partition. However, DPS reconfiguration is most commonly used by an operator to switch units from tbe green partition to tbe amber partition for M&DSS testing. The most commonly used system-error response is DPS recovery. It is the easiest to use because errors do not have to be localized beforehand. It is also the only system error response which may be invoked either by hardware (the sanity timer), by softwarc (the process coordinator), or manually by a DPS operator.

DPS recovery reinitializes the entire hardware/software system in approximately 10 to 20 seconds, depending on the CLC configuration size. Once initiated, DPS recovery proceeds automatically under the control of tbe operating system. It involves the following steps:

(*i*) Terminating process execution.
(*ii*) Saving the system image (including the data base, the contents of the status unit, and the contents of the processor registers) on disc for possible later analysis.
(*iii*) Running normal path diagnostics, and reconfiguring the CLC to eliminate faulty units if necessary.
(*iv*) Completely reinitializing the software by reloading all programs and the entire data base with fresh copies from disc.
(*v*) Resuming tactical execution.

VI. EXPERIENCE/RECOMMENDATIONS

The following are a few key points and recommendations based on the SAFEGUARD experience witb error control. The recommendations are believed to be generally applicable to designing error control into large-scale, real-time control systems.

(*i*) A system's error-control guidelines and error-control structure must be defined early. They are required early in the design if the system is to have a consistent approach to error control.
(*ii*) Error logging must be provided as one of thc first software functions. It is an invaluable debugging tool.
(*iii*) Certain error-control features, e.g., audits, must be considered early to make implementation feasible. SAFEGUARD might have made greater use of data-base audits if tbe data base had been designed with audits in mind.
(*vi*) Tcsting local error responses is difficult, but it is important for reliable operation. To enhance reliability, keep local responses simple and testable. To help simplify testing and to help reduce the amount of code devoted to local responses, categorize errors to minimize the number of different local

responses required. Many natural opportunities for testing local-error responses occur during early software testing. To take advantage of these opportunities, local-error responses must he implemented during early software development.

(*v*) Error responses should he easily modifiable. The desired responses may change as operational experience with a new system provides additional information about error occurrence rates. In the SAFEGUARD system, centralized, table-driven error-thresholding functions and system error-response maps permitted tailoring many of the local and system error responses as experience with the system grew.

(*iv*) Hardware and software status returns should he "response oriented." They should include a simple code indicating what to do ahout an error, that is: retry the operation; reset the device or correct a parameter first, then retry; don't retry, the device is broken; etc. More detailed status information to further identify the nature or cause of the error may also be included, but it should be independent of the response-oriented status. The detailed status may be recorded by software for off-line analysis.

(*vii*) Manual error control or manual override should be provided even for automatically operating or self-repairing systems. Manual control is essential for "bringing up" systems—even automatic systems. It is also invaluahle when automatic systems fail to operate, or when self-repairing systems fail to repair themselves.

REFERENCES

1. P. H. Enslow, Jr., Ed., *Multiprocessors and Parallel Processing*, New York: John Wiley, 1974.
2. A. K. Phillips, "SAFEGUARD Data-Processing System: Debugging a Real-Time Multiprocessor System," B.S.T.J., this issue, pp. S133–S145.
3. B. P. Donohue III and J. F. McDonald, "SAFEGUARD Data-Processing System: Process-System Testing and the System Exerciser," B.S.T.J., this issue, pp. S111–S122.
4. J. R. Hahn, Jr. and F. E. Slojkowski, "SAFEGUARD Data-Processing System: Maintenance and Diagnostic Subsystem," B.S.T.J., this issue, pp. S63–S72.